Hausarbeit

Beispieltitel für einen Bericht

Autor:	Vorname Nachname
Matrikel-Nr.:	xxxxxxx
Studiengang:	Maschinenbau
Erstprüfer:	Prof. Dr. Elmar Wings
Abgabedatum:	18. Juni 2021

Hochschule Emden/Leer · Fachbereich Technik · Abteilung Maschinenbau
Constantiaplatz 4 · 26723 Emden · `http://www.hs-emden-leer.de`

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Acronyms

1. Hinweise

Bitte verwenden Sie `biber.exe`.

Wenn Sie ein Symbolverzeichnis erstellen möchten, rufen Sie makeindex auf:
`makeindex %.nlo -s nomencl.ist`

2. Bézier-Kurven

2.1. Einleitung

Bézier-Kurven sind Parameterkurven, mit deren Hilfe Freiformkurven dargestellt werden können. Sie sind nach dem französischen Ingenieur Pierre Bézier benannt, der diese in den 1960er Jahren bei Renault entwickelte um Karosserieformen zu designen. Im selben Zeitraum entwickelte unabhängig von Pierre Bézier auch der französische Physiker und Mathematiker Paul de Casteljau diese Kurven bei Citroën. Paul de Casteljaus Ergebnisse lagen früher vor, doch wurden sie nicht[12] veröffentlicht. Dies ist der Grund dafür, dass Pierre Bézier Namensgeber dieser Parameterkurven ist.[**Farin:2002**]

Die Bézier-Kurven sind die Grundlage für das rechnerunterstützte Entwerfen von Modellen. Dieses wird entweder als Computer Aided Design (CAD) oder, wenn der Schwerpunkt mehr auf einer mathematischen Sicht liegt, als Computer Aided Geometric Design (CAGD) bezeichnet. [**Babovsky:2011**] Computer benötigt zum Darstellen von Formen eine mathematische Beschreibung dieser. Die am besten hierfür geeignete Beschreibungsmethode ist die Verwendung von parametrischen Kurven und Flächen. Hier spielen Bézier-Kurven die zentrale Rolle, denn sie sind die numerisch stabilsten Polynombasen die bei CAD/CAGD Software zum Einsatz kommen.[**Farin:2002**]

Bei der Typografie am Computer werden ebenfalls Bézier-Kurven genutzt. Es gibt zwei Arten Schrift mit dem Computer darzustellen. Die einfachste Art besteht darin, jeden einzelnen Buchstaben mit einer festen Auflösung und Größe zu als Bitmap zu speichern und bei Bedarf in den Speicherbereich des Bildschirms zu kopieren. Diese sogenannten Bitmap-Fonts sind so zwar schnell darstellbar aber benötigen viel Speicherplatz wenn die Schriftzeichen auch in verschiedenen Größen zur Verfügung stehen sollen. Hier muss eine extra Bitmap für jede Größe angelegt werden. Ebenfalls sinkt die Qualität wenn diese Bitmap-Fonts in Größe und Auflösung skaliert werden. Eine Alternative stellen hier die Vektor-Fonts da. Ihr Name leitet sich daher ab, dass sie in einem zweidimensionalen Vektorraum definierte Kurven zur Darstellung der Schriftzeichen verwenden. Der Vorteil liegt hier darin, dass die so dargestellten Schriftzeichen sich ohne Qualitätsverlust skalieren lassen. Bei den Vektor-Fonts haben sich zwei Standards entwickelt. Zu einem die TrueType-Fonts und zum anderen die PostScript-Fonts. Die TrueType-Fonts verwenden quadratische Bézier-Kurven während die PostScript-Fonts kubische Bézier-Kurven verwenden. Die kubischen Bézier-Kurven haben mehr Kontrollpunkte was zu eine besseren Qualität führt.[**Malaka:2009**]

Ein weiteres Anwendungsgebiet liegt in der Steuerung von Werkzeugmaschinen. Beim Abfahren einer Ecke muss die Achse am Eckpunkt bis zum Stillstand abgebremst und anschließend nach Richtungskorrektur wieder beschleunigt zu werden. Diese Vorgehen sorgt dafür, dass nicht mit einer konstanten Geschwindigkeit gefahren kann, was negative Folgen für die Zykluszeit und Qualität hat. Eine mögliche Lösung dieses Problems liegt darin statt einer scharfen Ecke eine Kurve zwischen die zwei Teilstrecken zu legen, die mit konstanter Geschwindigkeit abgefahren werden kann. Für diesen Einsatzzweck eigenen sich Bézier-Kurven, da nur zwei Punkte sowie zwei Tangenten benötigt werden um sie zu bilden. Bei einer Ecke lägen die Punkte sowie die Tangenten auf den Teilstrecken die die Ecke bilden. Der entstehende Fehler wäre analytisch kontrollierbar und würde eine Einstellung der Kurventoleranz erlauben.[**Sencer:2014**]

2.2. Allgemeine mathematische Beschreibung Bézier-Kurve

Bézier-Kurven werden mit Hilfe von Bernsteinpolynomen gebildet. Wenn mindestens zwei Punkte sowie zwei Tangenten bekannt sind, können Kontrollpunkte ermittelt werden mit denen ein Kontrollpolygon gebildet wird. An diesem Kontrollpolygon orientiert sich der Verlauf der Kurve. Die Anzahl der Kontrollpunkte ist abhängig vom Grad der Bézier-Kurve. Eine Bézier-Kurve vom Grad n hat n+1 Kontrollpunkte. Die Berechnung der Bézier-Kurve erfolgt über den De-Casteljau-Algorithmus.

Definition. Im Intervall [0; 1] ist das **Bernstein-Polynom n-ten Grades** definiert durch:

$$b_{i,n}(\lambda) = \binom{n}{i}(1-\lambda)^{n-i}\lambda^i, \quad \lambda \in [0,1], \quad i = 0,\ldots,n \tag{2.1}$$

Definition. Der Binomialkoeffizient ist definiert durch:

$$\binom{n}{i} = \frac{n!}{i!(n-i)!}, \quad i = 0,\ldots,n \tag{2.2}$$

Hierbei bilden die Bernstein-Polynome $b_{i,n}$ eine Basis des Vektorraumes $\mathbb{P}^n(I)$ der Polynome höchstens vom Grad n über I. Somit lässt sich jedes Polynom höchstens n-ten Grades eindeutig als Linearkombination schreiben. [**Farin:2002**]

Beispiel. Bestimmung von Bernstein-Polynome für $n = 3$ gilt:

$$\begin{aligned}
b_{3,0}(\lambda) &= \binom{3}{0}(1-\lambda)^{3-0}\lambda^0 &&= (1-\lambda)^3 \\
b_{3,1}(\lambda) &= \binom{3}{1}(1-\lambda)^{3-1}\lambda^1 &&= 3\lambda(1-\lambda)^2 \\
b_{3,2}(\lambda) &= \binom{3}{2}(1-\lambda)^{3-2}\lambda^2 &&= 3\lambda^2(1-\lambda) \\
b_{3,3}(\lambda) &= \binom{3}{3}(1-\lambda)^{3-3}\lambda^3 &&= \lambda^3
\end{aligned}$$

$b_{3,i}(\lambda)$ mit $i = 0,1,2,3$ sind die kubischen Bernstein-Polynome von Grad 3.

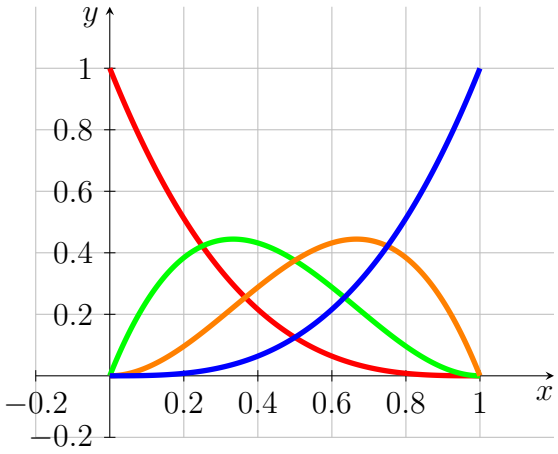


Abbildung 2.1.: Bernstein-Polynom vom Grad 3

Definition. Gegeben seien die Punkte

$$Q_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix} \quad \text{mit} \quad Q_i \in \mathbb{R}^2, \quad i = 0, 1, \ldots, n$$

Eine **Bézier-Kurve** ist dann definiert durch

$$C(\lambda) = \sum_{i=0}^{n} Q_i \cdot b_{i,n}(\lambda) \tag{2.3}$$

Die Punkte $Q_i, i = 0, \ldots, n$ heißen **Kontrollpunkte**.

Die Kontrollpunkte einer Bézier-Kurve bilden das sogenannte Kontrollpolygon.

Bemerkung. Es sei der Startpunkt P_0 und der Endpunkt P_1, sowie die Tangenten $\vec{t_0}$ und $\vec{t_1}$ gegeben. Die Tangenten sind nicht notwendigerweise normiert.
Die Kontrollpunkte Q_0, Q_1, Q_2 und Q_3 der zugehörigen Bézier-Kurve sind durch folgende Gleichungen gegeben:

$$Q_0 = P_0, \qquad Q_1 = P_0 + \lambda_0 \vec{t_0}, \qquad Q_2 = P_1 - \lambda_1 \vec{t_n}, \qquad Q_3 = P_1 \tag{2.4}$$

[**Jaklic:2010**]

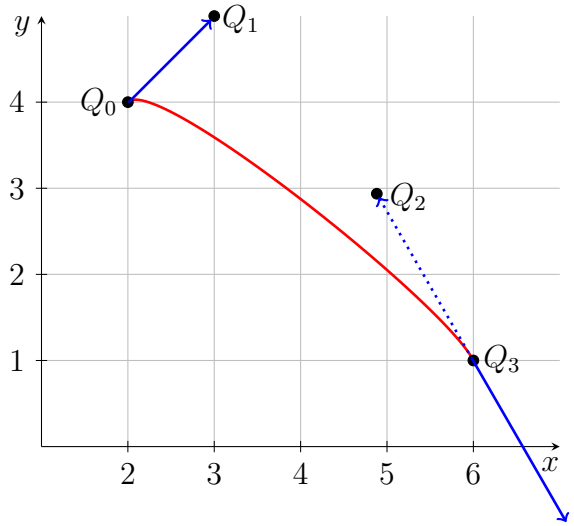


Abbildung 2.2.: Bézier-Kurve zum Beispiel 2.2

Beispiel. Gegeben seien

$$P_0 = \begin{pmatrix} 2 \\ 4 \end{pmatrix}, \quad \vec{t}_0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \text{und} \quad P_1 = \begin{pmatrix} 6 \\ 1 \end{pmatrix}, \quad \vec{t}_1 = \begin{pmatrix} 1 \\ -2 \end{pmatrix}$$

Dann ergibt sich gemäß Satz 2.4 für Kontrollpunkte der zugehörigen Bézier-Kurve:

$$Q_0 = P_0 = \begin{pmatrix} 2 \\ 4 \end{pmatrix}, \qquad Q_1 = P_0 + \vec{t}_0 = \begin{pmatrix} 2 \\ 4 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \end{pmatrix},$$

$$Q_2 = P_1 - \vec{t}_n = \begin{pmatrix} 6 \\ 1 \end{pmatrix} - \begin{pmatrix} 1 \\ -2 \end{pmatrix} = \begin{pmatrix} 5 \\ 3 \end{pmatrix}, \qquad Q_3 = P_1 = \begin{pmatrix} 6 \\ 1 \end{pmatrix}$$

Die Bézier-Kurve und ihre Kontrollpunkte sind in der Abbildung 2.2 dargestellt.

3. Verrundung mit einem Kreisbogen

3.1. Gleichungen

Die Verrundung mit einem Kreisbogen ist eine einfache Variante der Glättung von Ecken. Diese Variante ermöglicht die Bearbeitung und die Erzeugung von Dateien gemäß DIN 66025. Zur Glättung werden stets drei Punkte P_0 und S und P_1 betrachtet, damit ergibt sich ein symmetrisches Hermite-Problem. Gemäß Satz **??** wird vorausgesetzt, dass es sich in der Standardform $HP(L, \alpha)$ befindet.

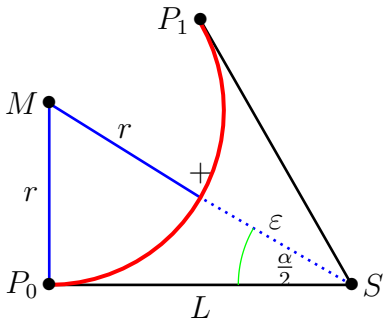


Abbildung 3.1.: Glättung einer Ecke mit Hilfe eines Kreisbogens - Dreieck

Die Abbildung 3.1 stellt die Situation dar. Die Punkte P_0, S und P_1 sind gegeben. Der eingeschlossene Winkel $\alpha = \angle(P_0; S; P_1)$ sowie der Abstand $L = |||S - P_0||| = |||P_1 - S|||$ sind in der Grafik dargestellt. Der rote Kreisbogen ist das gewünschte Ergebnis. Der Abstand seines Mittelpunkts M zu S beträgt dann $r + \varepsilon$, wobei r der Radius des Kreisbogens und ε die vorgegebene Toleranz ist. Somit erhalten wir ein rechtwinkliges Dreieck P_0SM, dessen Kantenlängen L, $r + \varepsilon$ und r sind.

Gemäß der Definition des Sinus und des Tangens folgt:

$$\sin\left(\frac{\alpha}{2}\right) = \frac{L}{r+\varepsilon} \qquad \text{und} \qquad \tan\left(\frac{\alpha}{2}\right) = \frac{L}{r}$$

$$\Leftrightarrow \quad \varepsilon = \frac{L}{\sin\left(\frac{\alpha}{2}\right)} - r \qquad \text{und} \qquad r = \cot\left(\frac{\alpha}{2}\right) L$$

Die beiden Gleichung können zusammengefasst werden, so dass sich folgende Bedingungen ergeben:

$$\varepsilon = L \cdot \frac{1-\cos\left(\frac{\alpha}{2}\right)}{\sin\left(\frac{\alpha}{2}\right)} \qquad \text{bzw.} \qquad L = \varepsilon \cdot \frac{\sin\left(\frac{\alpha}{2}\right)}{1-\cos\left(\frac{\alpha}{2}\right)}$$

Damit ergibt sich für den Radius die Gleichung:

$$r = L \cdot \cot\left(\frac{\alpha}{2}\right) = L \cdot \sqrt{\frac{1-\cos(\alpha)}{1+\cos(\alpha}} = L \cdot \frac{\sin(\alpha)}{1+\cos(\alpha)} = L \cdot \frac{P_{1,x}}{P_{1,y}}$$

Der Faktor zum Umrechnung von L und ε kann mit Hilfe trigonometrischer Umformungen vereinfacht werden.

$$\frac{1-\cos\left(\frac{\alpha}{2}\right)}{\sin\left(\frac{\alpha}{2}\right)} = \frac{1-\sqrt{\frac{1-\cos(\alpha)}{2}}}{\sqrt{\frac{1+\cos(\alpha)}{2}}} = \frac{\sqrt{2}-\sqrt{1-\cos(\alpha)}}{\sqrt{1+\cos(\alpha)}} = \frac{\sqrt{1+\cos(\alpha)}-\sin(\alpha)}{1+\cos(\alpha)}$$

Durch den Vergleich mit dem symmetrischen Hermite-Problem in Standardform ergibt sich dann die folgende Notation:

$$\frac{1-\cos\left(\frac{\alpha}{2}\right)}{\sin\left(\frac{\alpha}{2}\right)} = \frac{\sqrt{P_{1,x}}-P_{1,y}}{P_{1,x}}$$

Die vorherigen Überlegungen werden nun im nachfolgenden Satz zusammengefasst.

Satz. Es sei ein symmetrisches Hermite-Problem $(P_0, \vec{t}_0, P_1, \vec{t}_1, S, L)$ gegeben. Ohne Einschränkung der Allgemeinheit liegt es in der Standardform

$$\left\{\begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}L+L\cdot\cos(\alpha)\\L\cdot\sin(\alpha)\end{pmatrix}, \begin{pmatrix}\cos(\alpha)\\\sin(\alpha)\end{pmatrix}, \begin{pmatrix}L\\0\end{pmatrix}, L\right\}$$

mit $L \in \mathbb{R}^{>0}$ und $\alpha \in (-\pi;\, \pi]$ vor.

Dann kann ein Kreisbogen gefunden werden, der die Punkte P_0 und P_1 verbindet, die gleichen Tangentenrichtungen in den beiden Punkten besitzt.

Für die Kreisbogen gilt:

$$r = L \cdot \left|\frac{P_{1,x}}{P_{1,y}}\right|; \quad \phi_0 = -\operatorname{sign}(\alpha) \cdot \frac{\pi}{2}; \quad \phi_1 - \phi_0 = \operatorname{sign}(\alpha) \cdot \pi - \alpha = \beta;$$

$$M = P_0 + \operatorname{sign}(\alpha) \cdot r \cdot \vec{t}_0^{\perp} = \operatorname{sign}(\alpha) \cdot r \cdot \begin{pmatrix}0\\1\end{pmatrix}$$

Aus der Vorgabe des Abstand L kann, wie in Satz 3.1 dargestellt, der maximale Fehler berechnet werden. Gemäß der Herleitung ist es auch möglich, den maximalen Fehler ε vorzugeben und daraus den maximalen Abstand L zu bestimmen.

Satz. Es sei ein symmetrisches Hermite-Problem $(P_0, \vec{t}_0, P_1, \vec{t}_1, S, L)$ gegeben. Ohne Einschränkung der Allgemeinheit liegt es in der Standardform

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} L + L \cdot \cos(\alpha) \\ L \cdot \sin(\alpha) \end{pmatrix}, \begin{pmatrix} \cos(\alpha) \\ \sin(\alpha) \end{pmatrix}, \begin{pmatrix} L \\ 0 \end{pmatrix}, L \right\}$$

mit $L \in \mathbb{R}^{>0}$ und $\alpha \in (-\pi; \pi]$ vor.

a) Es sei der maximale Fehler ε vorgegeben. Der maximale Abstand L, für den ein Kreisbogen gemäß Satz 3.1 existiert, der den Fehler berücksichtigt, ist gegeben durch:

$$L(\varepsilon, \alpha) = \varepsilon \cdot \frac{P_{1,x}}{\sqrt{P_{1,x}} - P_{1,y}} = \varepsilon \cdot \frac{L + L \cdot \cos(\alpha)}{\sqrt{L + L \cdot \cos(\alpha)} - L + L \cdot \cos(\alpha)}$$

b) Bei der Vorgabe des Abstands L ergibt sich der folgende, maximale Fehler:

$$\varepsilon(L, \alpha) = L \cdot \frac{\sqrt{P_{1,x}} - P_{1,y}}{P_{1,x}} = L \cdot \frac{\sqrt{L + L \cdot \cos(\alpha)} - L + L \cdot \cos(\alpha)}{L + L \cdot \cos(\alpha)}$$

Aus diesen Überlegungen ergeben sich Einschränkungen für den Einsatz dieser Strategie, die in der folgenden Bemerkung zusammengefasst sind.

Bemerkung. Folgende Bedingungen zum Anwenden der Strategie eines Kreises müssen erfüllt sein:

a)
$$P_0 \neq P_1$$

b)
$$\vec{t}_0 = -\vec{t}_1 \quad \Leftrightarrow \quad \alpha = 0$$

c)
$$\vec{t}_0 = \vec{t}_1 \quad \Leftrightarrow \quad \alpha = \pi$$

3.2. Bewertung

Die Verrundung mittels eines Kreisbogens führt zu einem stetigen Verlauf der Winkeländerung. Die Abbildung 3.2 stellt dies dar.

Durch die Verrundung mit einem Kreisbogen wird ist der Krümmungsverlauf der Kurve nicht stetig. Die Abbildung 3.3 zeigt, dass die Krümmung im Bereich der Strecken 0 ist und auf dem Kreisbogen konstant mit dem Wert $\frac{1}{r}$, wobei r der Radius des eingesetzten Kreisbogens ist. Aufgrund der Formel $a = \frac{v^2}{r}$ für eine

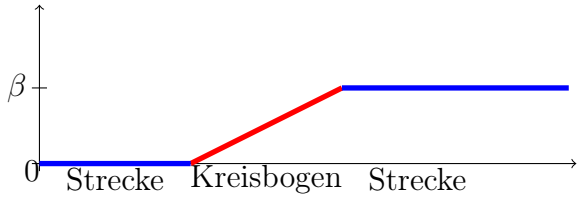


Abbildung 3.2.: Winkeländerung bei einer Verrundung mit einem Kreisbogen

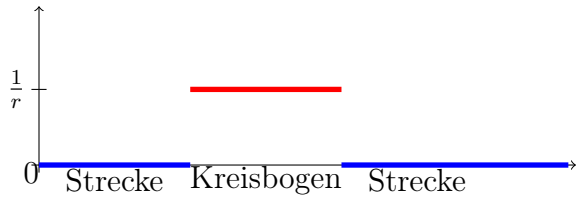


Abbildung 3.3.: Krümmungsverlauf bei der Verrundung mit einem Kreisbogen

Bewegung auf einem Kreisbogen weist der Beschleunigungsverlauf bei einer konstanten Bahngeschwindigkeit Stetigkeitssprüngen an den Übergängen aus.

In Abhängigkeit der Winkeländerung β an der Ecke S und der Toleranz ε kann die maximale Länge der Verkürzung der Strecken berechnet werden. Die Abbildung 3.4 zeigt das zugehörige Diagramm, wobei die Toleranz ε auf den Wert 1 gesetzt ist.

Der Bereich, der zur Verfügung gestellt wird, kann auch vorgegeben werden. In Abhängigkeit des Abstands L vom Eckpunkt S kann die Abweichung ε berechnet werden. Die Abbildung 3.5 stellt die Funktion in Abhängigkeit von L und des Winkels α dar.

Die Abbildungen 3.4 und 3.5 zeigen die Kurven der Länge in Abhängigkeit der Winkeländerung bei einer festen Toleranz und den Fehler in Abhängigkeit der Winkeländerung bei vorgegebenen Länge. Falls die Winkeländerung minimal ist, dann ist der Fehler oder die Länge L extrem. In diesem Fall ist eine Verrundung mittels eines Kreisbogens nicht möglich. Um eine sinnvolle Strategie zu entwickeln, muss eine

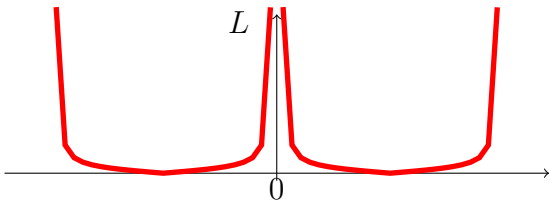


Abbildung 3.4.: Maximaler Bereich für die Verrundung mit einem Kreisbogen - $L(\varepsilon = const, \alpha)$

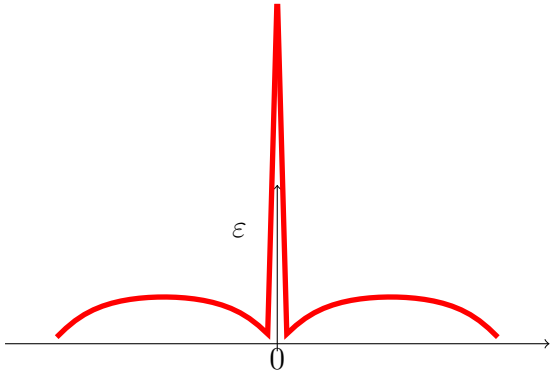


Abbildung 3.5.: Abweichung bei Vorgabe des Abstands L bei der Verrundung mit einem Kreisbogen

minimale Winkeländerung vorgegeben werden, ab der eine Verrundung mit einem Kreisbogen erlaubt ist. Ebenso ist eine maximale Winkeländerung zu definieren. Denn bei einer Winkeländerung von $\pm\pi$ handelt sich um eine Kehre. in diesem Fall ist eine Verrundung auch nicht möglich.

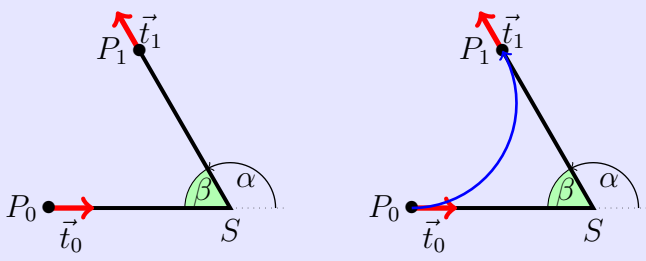
3.3. Beispiele

Beispiel. Es sei das symmetrische Hermite-Problem

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 6.0 + 6.0 \cdot \cos\left(\frac{2}{3}\pi\right) \\ 6.0 \cdot \sin\left(\frac{2}{3}\pi\right) \end{pmatrix}, \begin{pmatrix} \cos\left(\frac{2}{3}\pi\right) \\ \sin\left(\frac{2}{3}\pi\right) \end{pmatrix}, \begin{pmatrix} 6.0 \\ 0 \end{pmatrix}, 6.0 \right\}$$

mit $L = 6$ und $\alpha = \frac{2}{3}\pi$ gegeben.
Für den Verrundungskreisbogen folgt:

$$M = \begin{pmatrix} 0 \\ 3,4641 \end{pmatrix}; \quad r = 3,46412; \quad \phi_0 = -\frac{\pi}{2}; \quad \alpha = \frac{2}{3}\pi$$



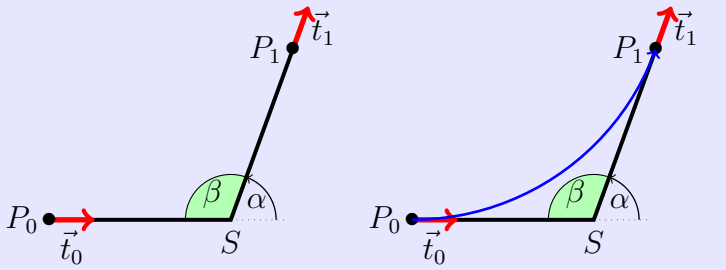
3. Verrundung mit einem Kreisbogen

Beispiel. Es sei das symmetrische Hermite-Problem

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 6.0 + 6.0 \cdot \cos\left(\frac{7}{18}\pi\right) \\ 6.0 \cdot \sin\left(\frac{7}{18}\pi\right) \end{pmatrix}, \begin{pmatrix} \cos\left(\frac{7}{18}\pi\right) \\ \sin\left(\frac{7}{18}\pi\right) \end{pmatrix}, \begin{pmatrix} 6.0 \\ 0 \end{pmatrix}, 6.0 \right\}$$

mit $L = 6$ und $\alpha = \frac{7}{18}\pi$ gegeben.
Für den Verrundungskreisbogen folgt:

$$M = \begin{pmatrix} 0 \\ 8,5689 \end{pmatrix}; \quad r = 8,5689; \quad \phi_0 = -\frac{\pi}{2}; \quad \alpha = \frac{7}{18}\pi$$

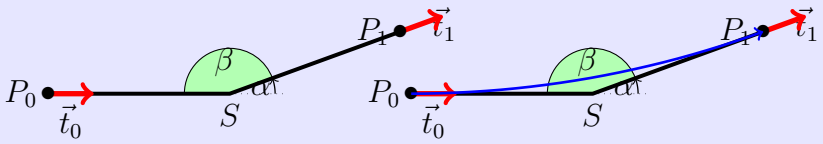


Beispiel. Es sei das symmetrische Hermite-Problem

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 6.0 + 6.0 \cdot \cos\left(\frac{1}{9}\pi\right) \\ 6.0 \cdot \sin\left(\frac{1}{9}\pi\right) \end{pmatrix}, \begin{pmatrix} \cos\left(\frac{1}{9}\pi\right) \\ \sin\left(\frac{1}{9}\pi\right) \end{pmatrix}, \begin{pmatrix} 6.0 \\ 0 \end{pmatrix}, 6.0 \right\}$$

mit $L = 6$ und $\alpha = \frac{1}{9}\pi$ gegeben.
Für den Verrundungskreisbogen folgt:

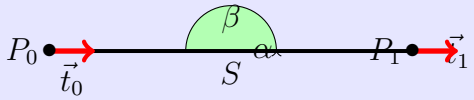
$$M = \begin{pmatrix} 0 \\ 34,0277 \end{pmatrix}; \quad r = 34,0277; \quad \phi_0 = -\frac{\pi}{2}; \quad \alpha = \frac{1}{9}\pi$$



Beispiel. Es sei das symmetrische Hermite-Problem

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 12.0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 6.0 \\ 0 \end{pmatrix}, 6.0 \right\}$$

mit $L = 6$ und $\alpha = 0$ gegeben.
Hier existiert kein Verrundungskreisbogen.

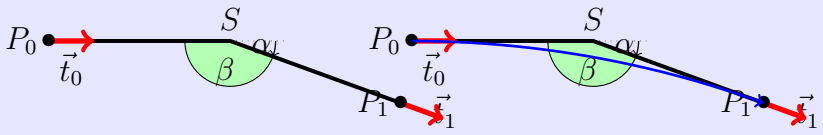


Beispiel. Es sei das symmetrische Hermite-Problem

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 6.0 + 6.0 \cdot \cos\left(-\frac{1}{9}\pi\right) \\ 6.0 \cdot \sin\left(-\frac{1}{9}\pi\right) \end{pmatrix}, \begin{pmatrix} \cos\left(-\frac{1}{9}\pi\right) \\ \sin\left(-\frac{1}{9}\pi\right) \end{pmatrix}, \begin{pmatrix} 6.0 \\ 0 \end{pmatrix}, 6.0 \right\}$$

mit $L = 6$ und $\alpha = -\frac{1}{9}\pi$ gegeben.
Für den Verrundungskreisbogen folgt:

$$M = \begin{pmatrix} 0 \\ -34,0277 \end{pmatrix}; \quad r = 34,0277; \quad \phi_0 = \frac{\pi}{2}; \quad \alpha = \frac{1}{9}\pi$$

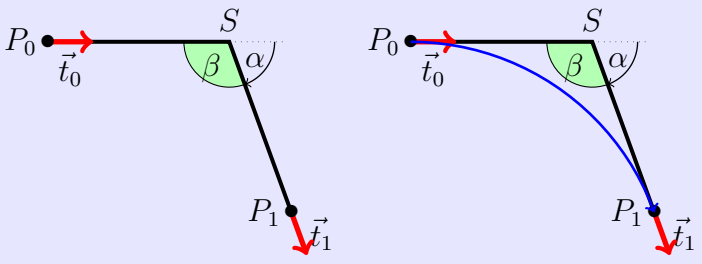


Beispiel. Es sei das symmetrische Hermite-Problem

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 6.0 + 6.0 \cdot \cos\left(-\frac{7}{18}\pi\right) \\ 6.0 \cdot \sin\left(-\frac{7}{18}\pi\right) \end{pmatrix}, \begin{pmatrix} \cos\left(-\frac{7}{18}\pi\right) \\ \sin\left(-\frac{7}{18}\pi\right) \end{pmatrix}, \begin{pmatrix} 6.0 \\ 0 \end{pmatrix}, 6.0 \right\}$$

mit $L = 6$ und $\alpha = -\frac{7}{18}\pi$ gegeben.
Für den Verrundungskreisbogen folgt:

$$M = \begin{pmatrix} 0 \\ -8,5689 \end{pmatrix}; \quad r = 8,5689; \quad \phi_0 = \frac{\pi}{2}; \quad \alpha = -\frac{7}{18}\pi$$

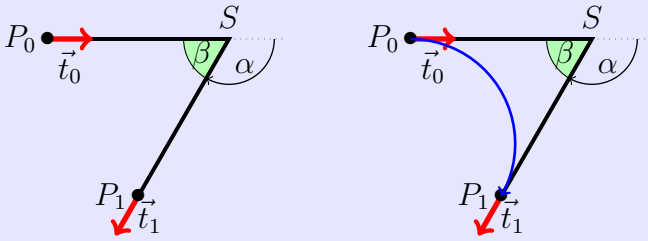


Beispiel. Es sei das symmetrische Hermite-Problem

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 6.0 + 6.0 \cdot \cos\left(-\frac{2}{3}\pi\right) \\ 6.0 \cdot \sin\left(-\frac{2}{3}\pi\right) \end{pmatrix}, \begin{pmatrix} \cos\left(-\frac{2}{3}\pi\right) \\ \sin\left(-\frac{2}{3}\pi\right) \end{pmatrix}, \begin{pmatrix} 6.0 \\ 0 \end{pmatrix}, 6.0 \right\}$$

mit $L = 6$ und $\alpha = -\frac{2}{3}\pi$ gegeben.
Für den Verrundungskreisbogen folgt:

$$M = \begin{pmatrix} 0 \\ -3,4641 \end{pmatrix}; \quad r = 3,46412; \quad \phi_0 = \frac{\pi}{2}; \quad \alpha = -\frac{2}{3}\pi$$



4. Maple-Dateien

[**Wat:2017a**; **Wat:2017b**; **Wat:2017c**; **Wat:2017d**; **Wat:2017e**; **Wat:2017f**]

4.1. Geometrien mit Maple

Die vorgestellten Algorithmen können gut mit Hilfe von Geometrien dargestellt werden. Dabei können zwei Aspekte untersucht werden. Einerseits ist der Aufwand für eine Umsetzung, die Rechengenauigkeit und die Stabilität eines Algorithmus interessant; andererseits sollen die Verfahren hinsichtlich ihres Nutzens bewertet und verglichen werden. Hierzu wurden Module für das Formelmanipulationssystem Maple [**Wat:2017a**] entwickelt.

Maple bietet die Umgebung, Algorithmen einfach und schnell zu implementieren. Darüber hinaus ist die grafische Darstellung mit einfachen Mitteln möglich. Allerdings ist eine einfache Arbeitsmappe von Maple nicht für sehr große Software-Projekte ausgelegt. Hier muss auf die Möglichkeit der Erstellung und Nutzung von Bibliotheken zurückgegriffen werden. Einerseits bietet Maple die Möglichkeit, eigene Bibliotheken, so genannte Module, zu erstellen. Auf diese Weise bleibt man innerhalb der Umgebung und der Syntax von Maple. Dieser Weg wird hier weiter verfolgt. Eine weitere Möglichkeit ist die Verwendung von Bibliotheken, die mittels einer höheren Programmiersprache, z.B. C++, erstellt wurde, den so genannten DLLs.

Im weiteren wird die Erstellung und Verwendung einer Testumgebung mit Hilfe von Module beschrieben. Zunächst werden die Geometrielemente beschrieben, die in verschiedenen Module abgelegt sind, und deren Verwendung. Damit eigene Erweiterungen und Ergänzungen möglich sind, wird dann der Aufbau und die Verwendung eines Moduls in Maple erläutert.

4.2. Verwendete Geometrieelemente

Da es sich um eine Testumgebung für die vorgestellten Algorithmen, werden auch nur Geometrien verwendet, die beschrieben wurden.

- Punkte (`MPoint`)
- Strecken (`MLine`)
- Kreisbögen (`MArc`)
- Bézier-Kurven (`Bezier`)

- Polygonzüge (`MPolygon`)
- Geometrieliste (`MGeoList`)
- Hermite-Probleme (`MHermiteProblem`)
- Symmetrische Hermite-Probleme (`MHermiteProblemSym`)

Für jedes Geometrieelement wurde ein entsprechendes Modul angelegt, dessen Name in der obigen Liste angegeben ist. Die Auflistung, welche Funktionen zur Verfügung stehen, wird in einem weiteren Abschnitt dargestellt.

Bei der Umsetzung eines solchen Projekts wird schnell deutlich, dass bei einer Verwendung von mehreren Geometrieelementen ein klare Datenstruktur und ein wohldefinierter Zugriff auf die Daten unerlässlich ist. Zum Beispiel bei der Darstellung von Geraden ist die Entscheidung zu treffen, ob die Darstellung Punkt-Richtung oder mittels zweier Punkte erfolgen soll. Die Auswahl erfolgt im Allgemeinen in Abhängigkeit des Anwendungsfalls. Hier wird der Weg beschritten, dass aufgrund eines wohldefinierten Zugriffs auf die Daten die Verwendung beider Darstellungen möglich ist.

4.3. Aufbau der Datenstruktur

Die Idee ist, eine möglichst einheitliche und einfache Datenstruktur zu verwenden. Maple bietet zwar die Möglichkeit, Objekte zu definieren. Allerdings ist die Verwendung innerhalb einer prozeduralen Umgebung schwierig. Daher werden alle Daten grundsätzlich als Listen dargestellt. Das erste Listenelement enthält dabei stets eine Kennung des Elements **??**. Dann folgen die Daten, die wiederum Geometrieelemente sein können. Es ist zu beachten, dass der direkte Zugriff auf die Daten nicht unterbunden werden kann; hier ist der Anwender in der Pflicht.

Der Zugriff auf die Daten soll ausschließlich über Prozeduren eines Moduls erfolgen. Im folgenden ist beispielhaft die Datenstruktur für Punkte zu sehen:

```
[MVPOINT,[x,y]]
```

Die Erstellung eines Punkts erfolgt dann über eine Prozedur `New`:

```
NeuerPunkt := MPoint:-New(10,15);
```

Beim Aufruf in der Testdatei wird dann folgendes ausgegeben:

```
TestPO := ["Point",[10,15]]
```

Diese Datenstrukturen werden für die Berechnung der Geometrien verwendet. Mit ihnen wird in Funktionen bzw. Prozeduren gearbeitet. Anders als Variablen werden die Datenstrukturen und Prozeduren hier global und nicht lokal definiert. Unter dem Befehl `export` werden die Prozeduren zu Beginn des Moduls deklariert und sind so auch außerhalb des Moduls verwendbar. Soll beispielsweise eine Datenstruktur aus

`MPoint` in einem anderen Modul oder außerhalb der Module verwendet werden wird sie wie folgt aufgerufen:

```
P0 := MPoint:-New(x,y);
```

Zusätzlich zu den Modulen für die Geometrien gibt es ein Modul (MConstant) zum Speichern von Konstanten und Namen. Dort ist für `MVPOINT` z.B. „Point “ gespeichert oder z.B. eine Konstante für den Vergleich auf Null für reelle Zahlen.

Zuletzt gibt es die Testdatei, welche kein Modul ist, in der die Module geladen, aufgerufen und in ihrer Funktion getestet werden. Zu dieser Datei später im Punkt „1.4 Maple Testdatei “ mehr.

4.4. Struktur eines Moduls

In dem folgenden Abschnitt geht es um den Zweck von Modulen und deren Struktur eingegangen.

Es geht in dem Projekt darum die oben genannten Geometrien in einer Datenstruktur zu erfassen und sie in Maple darzustellen. Für die letztendliche Darstellung sind die Berechnungen und Formeln, die verwendet werden müssen, allerdings hinderlich. Module eignen sich sehr gut dafür die Berechnungen, die in Funktionen erfolgen, zusammenzufassen und zu verstecken. So kann in Maple auf die wichtigen Funktionen zugegriffen werden, ohne dass man sieht, was in diesen steht.

Beispiel:
Die Funktion `Angle` aus dem Modul `MPoint`: Funktion zur Berechnung eines Winkels zwischen Ortsvektor und x-Achse:

```
Angle := proc(P)
    local alpha, x, y;
    x := GetX(P);
    y := GetY(P);
    alpha := 0;
    if abs(x) < 0.00001
        then
            alpha := 3*Pi/2;
        else
            alpha := Pi/2;
        end if;
    else
        alpha := arctan(y/x);
        if x < 0
        then
            alpha := alpha + Pi;
```

```
        else
            if y < 0
            then
                alpha := alpha + 2*Pi;
            end if;
        end if;
    end if;
    return factor(alpha);
end proc;
```

Diese Funktion ist lang, stellt aber nur einen kleinen Teil des Programms für die Darstellung dar, um die es geht. Deshalb steht sie in dem Modul und kann so extern mit einem einzigen Befehl aufgerufen werden:

```
Winkel := MPoint:-Angle(P)
```

Im Folgenden Abschnitt wird die Struktur eines Moduls beschrieben. Da Maple hier sehr vielfältige Möglichkeiten bietet, findet eine Beschränkung statt. Es wird nur die Struktur der verwendeten Module, hier anhand des Moduls`MPoint`, beschrieben. Für tiefergehende Möglichkeiten wird hier auf das Handbuch von Maple [**Wat:2017a**] verwiesen.

Struktur:

Das Modul muss zunächst gestartet werden. Dies erfolgt mit dem Namen (hier immer ein großes M und die Geometrie) des Moduls und dem folgenden Befehl:

```
MPoint := module()
```

Danach müssen, ähnlich wie in Prozeduren, die Variablen und Funktionen definiert werden. Dabei kann man entweder lokal oder global deklarieren. Werden die Variablen oder Prozeduren nur in dem Modul gebraucht und verändert, erfolgt die Deklaration mit dem Befehl `local` wie folgt:

```
local  Variablennamen, mit, Komma, getrennt;
```

Sollen die Variablen oder Prozeduren auch außerhalb des Moduls verwendbar sein, wird mit `export` dies definiert:

```
export Funktionsnamen, mit, Komma, getrennt;
```

Daraufhin folgt die Angabe der Optionen:
`option package;`,
die Beschreibung des Moduls:
`description “Selbstgewählte Modulbeschreibung z.B. Modul für Punkte “;`
und die Initialisierung des Moduls:

```
ModuleLoad := proc
   MVPOINT:=MConstant:-GetPoint();
   print("Modul MPoint ist geladen");
end proc;
```

Ab hier wird programmiert wie sonst in Maple auch. Man verwendet die vorher definierten Prozedur- und Variablennamen und programmiert mit Befehlen die man außerhalb eines Moduls in Maple auch verwendet. Wichtig zu wissen ist, dass bei Modulen jede Funktion ständig abgerufen werden kann, die Reihenfolge der Funktionen also keine Rolle spielt.

Um zuletzt das Modul zu beenden, nutzt man den folgenden Befehl:

```
end module;
```

4.4.1. Genereller Aufbau eines Moduls

Zusammenfassend haben die Module also folgendes Gerüst:

```
Modulname := module()

    local Namen, mit, Komma, getrennt;

    export Namen, mit, Komma, getrennt;

    global Namen, mit, Komma, getrennt;[1]

    option package;

    description "Selbstgewählte Modulbeschreibung";


    ModuleLoad := proc()[2]
        MVPOINT:=MConstant:-GetPoint();
        print("Modul MPoint ist geladen");
    end proc;

 Procedure1 := proc (Übergabeparameter)
        inhalt;
    end proc;

    Procedure2 := proc (Übergabeparameter)
```

[1] möglich, aber in den Modulen nicht verwendet
[2] Beispiel `MPoint`

```
        inhalt;
    end proc;


    ...

end module;
```

4.4.2. Sicherung eines Moduls

Die Verwaltung von Module wird von Maple automatisch durchgeführt. Da aber die Module weitergegeben werden und einzeln zu bearbeiten sind, müssen einige Einstellungen vorgenommen werden. Daher wird bei jeder Maple-Datei des Projekts folgender Präfix verwendet:

```
1 restart;
2 with(LibraryTools);
3 lib := "C:/FH/Tools/Maple/MyLibs/Blending.mla";
4 march('open', lib);
5 ThisModule := 'MArc';
```

Die 1.Zeile initialisiert das System. Die nächsten 3 Zeile ermöglichen das Arbeiten mit Archiven. In der zweiten Zeile werden die Werkzeuge geladen, so dass die Variable `lib` belegt werden kann. Alle Module werden in einem Archiv abgelegt; in diesem Beispiel ist es die Datei `Blending.mla` im Verzeichnis `C:/FH/Tools/Maple/MyLibs/`. Das Kommando `ThisModule := 'MArc';` enthält den Namen des aktuellen Moduls.

Ein Modul wird dann mittels des Befehls `savelib('ModuleName')` gespeichert. Es wird dann eine Datei `ModuleName.mla` angelegt. Je nach Konfiguration von Maple ist der eingestellte Pfad, wo die Datei automatisch angelegt wird, nicht beschreibbar. Dann kann man das System so konfigurieren, dass die mla-Datei im aktuellen Verzeichnis oder in einem Verzeichnis der eigenen Wahl abgelegt wird.

Falls der obige Vorspann für eine Maple-Datei verwendet wird, genügt zum Speichern des Moduls

```
savelib(ThisModule, lib);
```

4.4.3. Verwendung eines Moduls

Ein Modul, das abgespeichert wurde, kann nun in anderen Maple-Worksheets verwendet werden. Dazu wird der Befehl

```
with(ModuleName)
```

verwendet. Falls man einen speziellen Pfad verwendet hat, so kann Maple die Datei `ModuleName.mla` nicht finden. Dann muss der Pfad bekannt gemacht werden, z.B.:

```
savelibname := "c:/Maple/MyLibs";", savelibname;
```

Nun kann man auf die Prozeduren des Moduls zugegriffen werden. Eine Übersicht über alle exportierten Prozeduren wird durch den Befehl

```
Describe(ModuleName)
```

angezeigt. Dabei wird eine List der Namen inklusiver der Namen der Übergabeparameter dargestellt. Falls eine Prozedur mit der Feld `description` ausgestattet ist, so wird dieser Text zusätzlich wiedergegeben.

4.4.4. Erstellung eines Maple-Moduls

Die Erstellung eines eigenen Moduls ist schnellt erledigt, falls man den obigen Rahmen verwendet. Allerdings sollte man vorher einige Überlegungen anstellen und einen Standard pflegen.

Bevor ein eigenes Modul erstellt wird, sollte das Datenmodell und die zugehörigen Prozeduren erarbeitet sein. Ein Programmablaufplan leistet hier gute Dienste. Die zentrale Aufgabe des Moduls ist in der Regel schnell ermittelt. Zusätzlich sollten aber folgende Regeln eingehalten werden.

Regel 1 Grundsätzlich erhält sowohl das Modul als auch jede Prozedur eine Beschreibung. Diese beschränkt sich nicht auf das optionale Argument `description`, sonder wird jeder Prozedur vorangestellt. Die Beschreibung enthält grundsätzlich die Beschreibung der Aufgabe. Dabei werden auch die Voraussetzung genannt bzw. eine Fehlerbhandlung beschrieben. Dann folgt die Beschreibung aller Eingabeparameter, deren Funktion und deren Datenstruktur. Der Rückgabewert wird anschließend beschrieben.

Regel 2 Jedes Modul erhält eine Prozedur `Version()`, die die aktuelle Versionsnummer zurückliefert.

Regel 3 Daten sind nicht global. Um auf Daten zugreifen zu können, werden Proceduren `Set*` und `Get*` zur Verfügung gestellt. Auch innerhalb der Prozeduren eines Moduls werden diese Prozeduren verwendet.

Regel 4 Für jede Prozedur wird mindestens eine Testfunktion geschrieben. Anhand der Testfunktion kann die Verwendung und ggfs. Besonderheiten dargestellt werden.

4.5. Funktionen der Module

Im folgendem werden die einzelnen Module aufgeführt. Die Datenstruktur jedes Moduls und alle Funktionen mit zugehöriger Aufgabe werden genannt.

4.5.1. Modul `MPoint`

`MPoint` ist ein Modul für Punkte und arbeitet mit einer Datenstruktur und mit Prozeduren/Funktionen. Die Datenstruktur `New` für einen Punkt ist eine Liste, die wie folgt aufgebaut ist:

```
[MVPOINT, [x,y]]
```

Ihr erstes Element ist ein Name zur Kennung des Moduls. Ihr zweites Element ist erneut eine Liste in der sich die Elemente der Geometrie befinden. In diesem Fall ist der Name "Pointünd die Elemente sind die x- und die y-Koordinate für einen Punkt. Es wird hier kein Unterschied zwischen Punkt und Vektor gemacht. Ein Punkt kann auch als Vektor vom Koordinatenursprung zum Punkt gesehen werden.

Über die Get-Funktionen `GetX` und `GetY` kann man die Koordinaten des Punktes erfassen und in anderen Funktionen verwenden. Über die anderen Funktionen kann mit den Punkten/Vektoren dann gerechnet werden.

`MPoint`		
E	`New`	Datenstruktur für einen Punkt
E	`GetX`	Lesen der x-Koordinate
E	`GetY`	Lesen der y-Koordinate
E	`Angle`	Berechnung des Winkels zwischen x-Achse und dem Punkt
E	`Add`	Errechnet eine Linearkombination zweier Punkte/-Vektoren
E	`Sub`	Errechnet die Differenz zweier Punkte/Vektoren
E	`Cos`	Errechnet den Cosinus zwischen zwei Vektoren
E	`Sin`	Errechnet den Sinus zwischen zwei Vektoren
E	`Scale`	Skaliert einen Vektor mit einem Faktor
E	`Perp`	Errechnet einen Vektor, der orthogonal zum angegebenen Vektor ist
L	`IllustrateXY`	Plot Funktion zur Darstellung eines blauen Punktes
E	`Illustrate`	Darstellung eines blauen Punktes
E	`Plot`	Darstellung eines grünen Punktes
E	`Plot2D`	Darstellung eines Punktes mit eigenen Optionen
E	`Length`	Errechnet den Abstand des Punktes zum Koordinatenursprung
E	`Uniform`	Normiert den Vektor auf die Länge 1
E	`LinetoVector`	Errechnet Vektor aus einer Strecke
E	`Distance`	Errechnet den Abstand zwischen zwei Punkten

4.5.2. Modul `MLine`

`MLine` ist ein Modul für Strecken und arbeitet mit einer Datenstruktur und mit Prozeduren/Funktionen. Die Datenstruktur `New` für eine Strecke ist eine Liste, die wie folgt aufgebaut ist:

```
[MVLINE, [P0,P1]]
```

Ihr erstes Element ist ein Name zur Kennung des Moduls. Ihr zweites Element ist erneut eine Liste in der sich die Elemente der Geometrie befinden. In diesem Fall ist der Name „Line“ und die Elemente sind der Start- und der Endpunkt für eine Strecke.

Die Datenstruktur `NewPointVerctor` ist ebenfalls eine Datenstruktur für eine Strecke, allerdings wird die Strecke hier über einen Startpunkt und einen Richtungsvektor definiert.

Über die Get-Funktionen StartPoint und EndPoint kann man Start- und Endpunkt der Strecke erfassen und in anderen Funktionen verwenden. Über die anderen Funktionen kann mit den Strecken dann gerechnet werden.

MLine		
E	New	Datenstruktur für eine Strecke (Zwei-Punkt-Form)
E	NewPointVector	Schaffung einer Strecke (Punkt-Richtungs-Form)
E	StartPoint	Lesen des Startpunktes
E	EndPoint	Lesen des Endpunktes
E	Position	Errechnen eines Punktes der auf der Strecke liegt
E	Plot2D	Darstellung eines Teils der Strecke ausgehend vom Startpunkt
E	Plot2DTangent	Darstellung eines Punktes mit Tangente
E	Plot2DTangentArrow	Darstellung eines Punktes mit Tangente (als Pfeil)
E	LineLine	Errechnung des Schnittpunktes zweier Geraden
E	AngleLineLine	Errechnung des Winkels zwischen zwei Geraden

4.5.3. Modul MArc

MArc ist ein Modul für Kreisbögen und arbeitet mit einer Datenstruktur und mit Prozeduren/Funktionen. Die Datenstruktur New für einen Kreisbogen ist eine Liste, die wie folgt aufgebaut ist:

```
[MVARC, [mx,my,r,phi,alpha]]
```

Ihr erstes Element ist ein Name zur Kennung des Moduls. Ihr zweites Element ist erneut eine Liste in der sich die Elemente der Geometrie befinden. In diesem Fall ist der Name Ärcünd die Elemente sind die x- und y-Koordinate für den Mittelpunkt, der Radius, Der Startwinkel und die Winkeländerung für einen Kreisbogen.

Über die Get-Funktionen GetM, GetMX, GetMY, GetR, GetPhi, und GetAlpha kann man die Daten für einen Kreisbogen erfassen und in anderen Funktionen verwenden. Über die anderen Funktionen kann mit den Kreisbögen dann gerechnet werden.

MArc

E	New	Datenstruktur für einen Kreisbogen
E	GetMX	Lesen der x-Koordinate des Mittelpunktes
E	GetMY	Lesen der y-Koordinate des Mittelpunktes
E	GetR	Lesen des Radius
E	GetPhi	Lesen des Startwinkels
E	GetAlpha	Lesen der Winkeländerung
E	GetM	Lesen des Mittelpunktes
E	Position	Errechnung eines Punktes auf dem Kreisbogen
E	Plot2D	Darstellung eines Teils des Kreisbogens ausgehend vom Startwinkel
E	Blend	Errechnung eines Kreisbogens aus einem symmetrischen Hermite-Problem

4.5.4. Modul MBezier

Die Datenstruktur New für ein Polygon ist eine Liste, die wie folgt aufgebaut ist:

```
[MVBEZIER, [PointList]]
```

Innerhalb des Modules MBezier existieren die in folgender Tabelle aufgeführten Prozeduren.

MBezier

E	New	Manuelle Eingabe von Kontrollpunkten für eine Bézier-Kurve
E	Version	Ausgabe der Verions
E	BlendCurvature	Bestimmung von Kontrollpunkten aus symmetrischen Hermite-Problem
E	BlendCurvatureEpsilon	Bestimmung von Kontrollpunkten aus symmetrischen Hermite-Problem mit vorgegebenen Fehler
E	Position	Position auf der Bézier-Kurve
E	GetTheta	Lesen des Winkels
E	GetEpsilon	Lesen des Toleranz
E	GetControlPoint	Lesen der Kontrollpunkte
E	Plot2D	Darstellung der Bézier-Kurve
E	PlotControlPoints	Darstellung aller Kontrollpunkte

4.5.5. Modul MPolygon

MPolygon ist ein Modul für Polygonzüge und arbeitet mit einer Datenstruktur und mit Prozeduren/Funktionen. Die Datenstruktur New für ein Polygon ist eine Liste, die wie folgt aufgebaut ist:

```
[MVPOLYGON, [PointList]]
```

Ihr erstes Element ist ein Name zur Kennung des Moduls. Ihr zweites Element ist erneut eine Liste in der sich die Elemente der Geometrie befinden. In diesem Fall ist der Name "Polygonünd die Elemente sind beliebig viele Punkte.

Über die Get-Funktionen `GetPoint` und `GetN` kann man die Anzahl der Punkte und die Punkte selbst erfassen und in anderen Funktionen verwenden. Über die anderen Funktionen kann mit den Punkten bzw. dem Polygonzug dann gerechnet werden.

MPolygon		
E	`New`	Datenstruktur für eine Punkteliste
E	`GetPoint`	Lesen des i-ten Punktes aus der Punkteliste
E	`GetN`	Bestimmung der Anzahl der Punkte in der Punkteliste
E	`Length`	Bestimmung der euklidischen Länge des Polygonzuges
E	`Position`	Berechnung eines Punktes auf dem Polygonzug
E	`Tangents`	Berechnung der Tangenten
L	`Plot2DAll`	Plotliste aller Punkte
E	`Plot2D`	Darstellung aller Punkte als Polygonzug
E	`Plot2DTangent`	Darstellung des Polygonzuges mit Tangenten
E	`PlotPoints`	Darstellung aller Punkte

4.5.6. Modul `MGeoList`

`MGeoList` ist ein Modul für eine Geometrieliste und arbeitet mit einer Datenstruktur und mit Prozeduren/Funktionen. Die Datenstruktur `New` für eine Geometrieliste ist eine Liste, die wie folgt aufgebaut ist:

```
[MVGEOLIST, []]
```

Ihr erstes Element ist ein Name zur Kennung des Moduls. Ihr zweites Element ist erneut eine Liste in der sich die Elemente der Geometrie befinden. In diesem Fall ist der Name „GeoList“ und die Elemente sind beliebig viele einzelne Geometrien.

Über die Get-Funktionen `GeoGeo` und `GetN` kann man das i-te Element der Liste und die Anzahl der Elemente in der Liste erfassen und in anderen Funktionen verwenden. Über die anderen Funktionen kann mit den Geometrien bzw. der Liste dann gerechnet werden. In den Funktionen `Length`, `Plot2DAll`, `Plot2D` und `Position` werden die Funktionen in sich selber aufgerufen. Dies ist möglich da die Funktionen unabhängig voneinander funktionieren.

MGeoList		
E	New	Datenstruktur für eine Geometrieliste
E	Append	Anfügen eines Geometrieelements in die Datenstruktur
E	Prepend	Einfügen eines Geometrieelements als erstes Element der Liste
E	Replace	Ersetzen eines Geometrieelements durch ein Anderes
E	GetN	Bestimmung der Anzahl der Geometrieelemente in der Liste
E	GeoGeo	Lesen des i-ten Geometrieelements
E	Length	Errechnet die euklidische Länge der Geometrieelemente
E	Position	Berechnung eines Punktes auf der Geometrie
L	Plot2DAll	Plot Funktion für die Darstellung der Geometrieelemente
E	Plot2D	Darstellung eines Teils der Geometrieliste ausgehend vom ersten Element

4.5.7. Modul MHermiteProblem

MHermiteProblem ist ein Modul für Hermite-Probleme und arbeitet mit einer Datenstruktur und mit Prozeduren/Funktionen. Die Datenstruktur New für eine Strecke ist eine Liste, die wie folgt aufgebaut ist:

```
[MVHERMITEPROBLEM, [P0,T0n,P1,T1n]]
```

Ihr erstes Element ist ein Name zur Kennung des Moduls. Ihr zweites Element ist erneut eine Liste in der sich die Elemente der Geometrie befinden. In diesem Fall ist der Name „HermiteProb“ und die Elemente sind zwei Punkte mit zugehörigen Tangenten (Strecken).

Über die Get-Funktionen StartPoint, EndPoint, StartTangent und EndTangent kann man die Punkte und ihre Tangenten erfassen und in anderen Funktionen verwenden. Über die anderen Funktionen kann mit den Daten für das Hermite-Problem dann gerechnet werden.

MHermiteProblem		
E	New	Datenstruktur für ein Hermite-Problem
E	StartPoint	Lesen des Startpunktes
E	EndPoint	Lesen des Endpunktes
E	StartTangent	Lesen der Starttangente
E	EndTangent	Lesen der Endtangente
E	Plot2D	Darstellung des Hermite Problems

4.5.8. Modul MHermiteProblemSym

MHermiteProblemSym ist ein Modul für symmetrische Hermite-Probleme und arbeitet mit einer Datenstruktur und mit Prozeduren/Funktionen. Die Datenstruktur New für ein symmetrisches Hermite-Problem ist eine Liste, die wie folgt aufgebaut ist:

```
[MVHERMITEPROBLEMSYM, [P0,T0n,P1,T1n,S,L]]
```

Ihr erstes Element ist ein Name zur Kennung des Moduls. Ihr zweites Element ist erneut eine Liste in der sich die Elemente der Geometrie befinden. In diesem Fall ist der Name „SymHermiteProb“ und die Elemente sind zwei Punkte mit zugehörigen Tangenten, ihr Schnittpunkt, und der Abstand der Punkte zum Schnittpunkt.

Über die Get-Funktionen StartPoint, EndPoint, StartTangent, EndTangent, CrossPoint und ParameterL kann man die Daten erfassen und in anderen Funktionen verwenden. Über die anderen Funktionen kann mit den Daten für das symmetrische Hermite-Problem dann gerechnet werden.

MHermiteProblemSym		
E	New	Datenstruktur für ein Symmetrisches Hermite-Problem
E	StartPoint	Lesen des Startpunktes
E	EndPoint	Lesen des Endpunktes
E	StartTangent	Lesen der Starttangente
E	EndTangent	Lesen der Endtangente
E	ParameterL	Lesen des Abstandes
E	CrossPoint	Lesen des Schnittpunktes
E	Plot2D	Darstellung des symmetrischen Hermite Problems
E	Create	Erzeugung eines Sym. Hermite-Problems durch 3 Punkte
E	BlendArc	Verrundung des Eckpunktes durch einen Kreisbogen

4.5.9. Modul MConstant

MConstant ist ein Modul zum Speichern von festen Konstanten und Namen zur Kennung der Datenstrukturen. In der lokal deklarierten Funktionen, beginnend mit CV, werden die Konstanten zur Deklaration der verschiedenen Geometrien definiert. Dies sind Namen zur Erkennung der Geometrieelemente in der Testdatei. In den global deklarierten Funktionen, beginnend mit Get, erfolgt die Rückgabe der Namen zur Kennung der Geometrieelemente.

MConstant		
L	NULLEPS	Konstante zum Vergleich auf Null
L	CVPOINT	Konstante für Geometrieelemente: Point
L	CVLINE	Konstante für Geometrieelemente: Line
L	CVARC	Konstante für Geometrieelemente: Arc
L	CVPOLYGON	Konstante für Geometrieelemente: Polygon
L	CVGEOLIST	Konstante für Geometrieelemente: GeoList
L	CVHERMITEPROBLEM	Konstante für Geometrieelemente: Hermite-Prob
L	CVHERMITEPROBLEMSYMMETRIC	Konst. für Geometrieelemente: SymHermite-Prob
L	CVBIARC	Konst. für Geometrieelemente: Biarc
E	GetNullEps	Rückgabe des Nullvergleichs
E	GetPoint	Kennung für Punkte
E	GetLine	Kennung für Strecken
E	GetArc	Kennung für Kreisbögen
E	GetPolygon	Kennung für Polygone
E	GetGeoList	Kennung für Geometrielisten
E	GetHermiteProblemSymmetric	Kennung für symmetrische Hermite-Probleme
E	GetHermiteProblem	Kennung für Hermite-Probleme
E	GetBiarc	Kennung für Biarcs

4.5.10. Modul MGeneralMath

MGeneralMath ist ein Modul für allgemeine mathematische Funktionen und arbeitet mit den Datenstrukturen und Prozeduren.

MGeneralMath		
E	MPoint	Datenstruktur für einen Punkt
E	MPointX	Lesen der x-Koordinate für einen Punkt
E	MPointY	Lesen der y-Koordinate für einen Punkt
L	MPointIllustrateXY	Plotstruktur für einen blauen Punkt
E	MPointIllustrate	Darstellung eines blauen Punktes
E	MPointPlot	Darstellung eines grünen Punktes
E	MLine	Datenstruktur für eine Strecke
E	MLineStartPoint	Lesen des Startpunktes für eine Strecke
E	MLineEndPoint	Lesen des Endpunktes für eine Strecke
E	MPointOnLine	Berechnung eines Punktes auf der Strecke
E	MLinePlot2D	Darstellung des Teils einer Strecke beginnend beim Startpunkt
E	MLineLine	Errechnen des Schnittpunktes zweier Strecken

4.5.11. Modul Biarc

Datenstruktur

Voraussetzungen und Festlegungen:

Der Biarc soll ein dafür verwendet werden ein Hermite-Problem zu lösen. Ein Hermite Problem ist wie folgt definiert:

Gegeben seien zwei Punkte P_0 und P_1 mit zugehörigen normierten Tangenten $\vec{t_0}$ und $\vec{t_1}$. Der Biarc muss die Punkte so verbinden, dass die Tangenten des Start- und des Endpunktes des Biarcs mit den Tangenten des Hermites-Problems übereinstimmen.

Datenstruktur:

Die Datenstruktur `New` für einen Biarc muss zwei Kreisbögen enthalten.

Benötigt wird also die Datenstruktur für einen Kreisbogen: `MArc:-New`

Diese enthält wiederum die Koordinaten für den Mittelpunkt, den Radius, den Startwinkel und die Winkeländerung.

4.5.12. Modul MBiarc

`MBiarc` ist ein Modul für Biarcs und arbeitet mit einer Datenstruktur und mit Prozeduren/Funktionen. Die Datenstruktur `New` für einen Biarc ist eine Liste, die wie folgt aufgebaut ist:

```
[MVBIARC, [Arc0, Arc1]]
```

Ihr erstes Element ist ein Name zur Kennung des Moduls. Ihr zweites Element ist erneut eine Liste in der sich die Elemente der Geometrie befinden. In diesem Fall ist der Name „Biarc“ und die Elemente sind zwei Kreisbögen.

Über die Get-Funktionen `GetArc0` und `GetArc1` kann man die beiden Kreisbögen für den Biarc erfassen. Mit den unten aufgeführten Funktionen kann man die Daten für den Biarc errechnen.

MBiarc		
E	New	Datenstruktur für einen Biarc
E	GetArc0	Lesen des ersten Kreisbogens
E	GetArc1	Lesen des zweiten Kreisbogens
E	Circle	Berechnung des Kreises K_J aus den Daten des Hermite-Problems
E	Plot2DCircle	Darstellung des Kreises K_J
E	angle	Errechnet den Winkel vom Kreismittelpunkt zu Punkten auf dem Kreis
E	Plot2D	Darstellung des Biarcs
E	ConnectionPoint	Errechnung des VerbindungsPunktes J (Equal Chord)
E	TangentTj	Errechnen der Tangente an J
E	Tangent Biarc	Drehung von Tj für den Biarc
E	BiarcCenter	Errechnen der Mittelpunkte der Kreisbögen des Biarcs
E	Biarc	Errechnen des Biarcs
E	Position	Ermitteln eines Punktes auf dem Biarc
E	Blend	Errechnen des Biarcs nur aus dem Hermite-Problem

4.6. Programmablaufplan

4.6.1. Gesamtablauf

4.6.2. Verrundung der Kurve

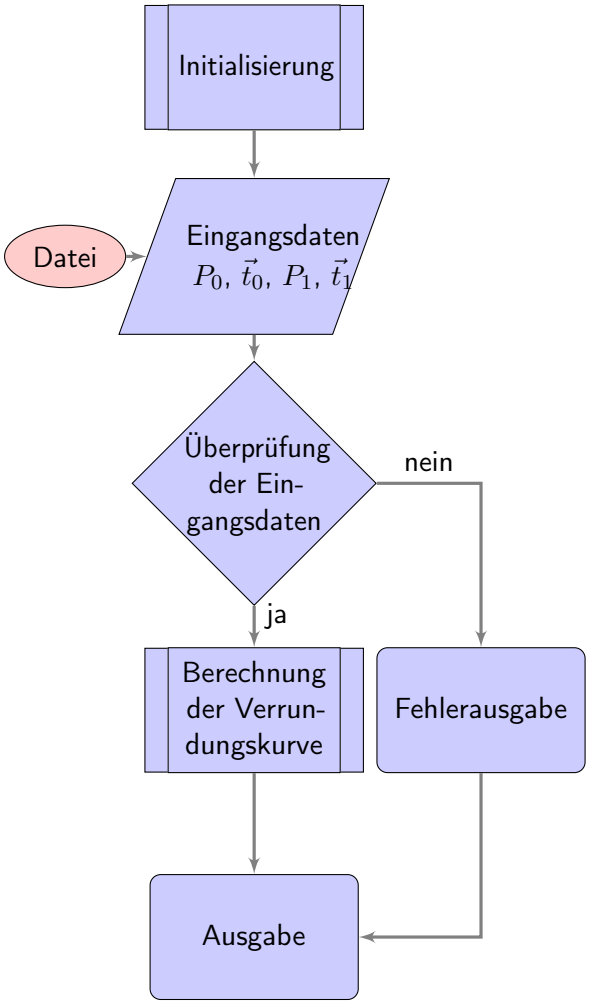


Abbildung 4.1.: Programmablaufplan „Eckenverrrundung“

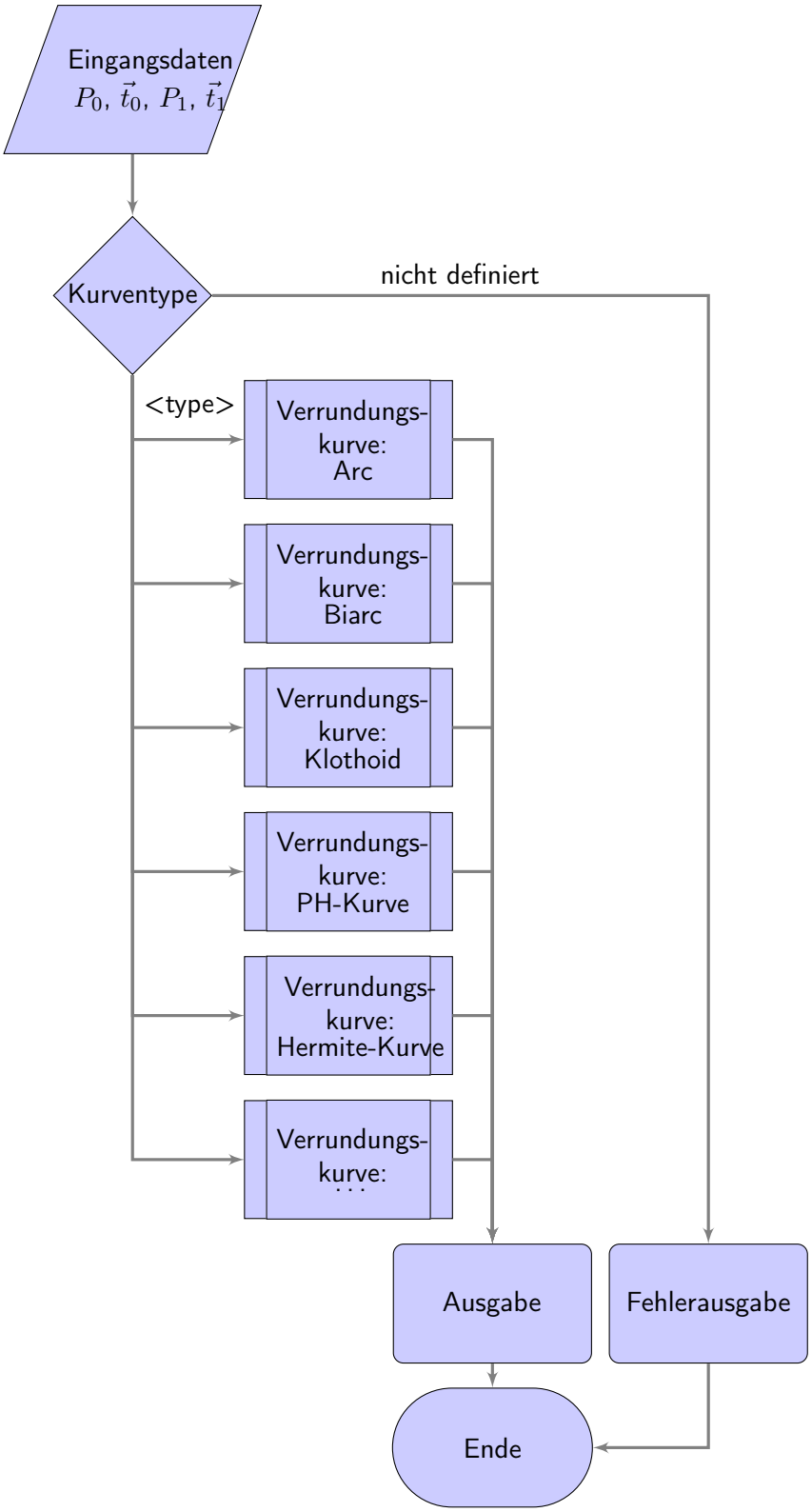


Abbildung 4.2.: Programmablaufplan „Auswal der Verrundungsstrategien“

5. Erstes Kapitel

...

Eine *Speicherprogrammierbare Steuerung* (SPS) ist ...

Eine *Computerized Numerical Control* (CNC) benötigt eine SPS (SPS) zur ...

6. CAGD

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Das Buch CAGD von Gerald Farin ist ein Klassiker über Splines. [**Farin:2002**]

Die Norm 66025 zur Programmierung von CNC-Maschinen ist ebenfalls ein Klassiker; sie behandelt allerdings keine Splines. [**DIN66025**]

Herr F. Farouki hat sich sowohl mit der Programmierung von CNC-Maschinen als auch mit Splines beschäftigt. Sein Artikel[1] über einen Echtzeitinterpolator zeigt dies auch. [**Farouki:2017**]

Ein neue Dimension der Werkzeugmaschinen sind durch die Erfindung von 3D-Drucker entstanden. [**Patent3D**]

Ein anderer Aspekt der Automatisierungstechnik ist die Kommunikation. Durch die 5G-Technik ist hier ein weiterer Meilenstein erreicht worden.[2]

[1] Mitautor ist J. Srinathu

[2] **Zafeiropoulos:2020**.

A. Zeichnungen mit tikz

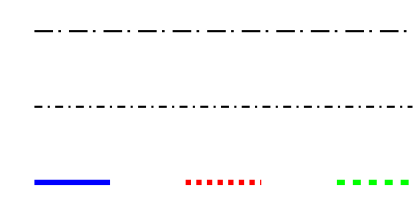
Das Paket tikz ist ein mächtiges Werkzeug zu erstellen von Grafiken. Es existieren vielen Einführungen. Hier werden nur die ersten Schritte gezeigt, so dass man leicht Flussdiagramme erzeugen kann.

Zeichnen einer Linie und Pfeile

```
\begin{tikzpicture}
    \draw (0,0) -- (1,0);

    \draw[->] (2,0) -- (3,0);

    \draw[<->] (5,0) -- (6,0);
\end{tikzpicture}
```

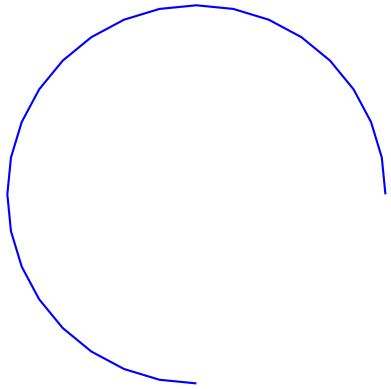

Zeichnen einer dicken, blauen Linie

```
\begin{tikzpicture}
    \draw[line width=2pt, blue] (0,0) -- (1,0);

    \draw[line width=2pt, red, dotted] (2,0) -- (3,0);

    \draw[line width=2pt, dashed, green] (4,0) -- (5,0);

    \draw [thick,dash dot] (0,1) -- (5,1);
    \draw [thick,dash pattern={on 7pt off 2pt on 1pt off 3pt}] (0,2) --
\end{tikzpicture}
```



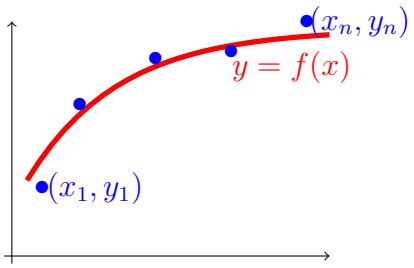
Zeichnen eines Kreisbogens

```
\begin{tikzpicture}
  \draw [blue,thick,domain=0:270] plot ({5+2.5*cos(\x)}, {1+2.5*sin(\x)
\end{tikzpicture}
```



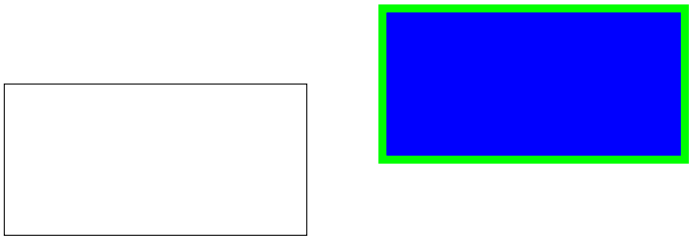
Zeichnen einer Funktion

```
\begin{tikzpicture}[
    declare function={%
        F(\x)                   =3-2*pow(2.7979,-0.8*\x);
    }
 ]


    % Zeichnen der Funktion
    \draw[red,line width=2pt,domain=0:4] plot({\x},{F(\x)});

    % Bezeichnung
    \node[red] (O) at(3.5,2.5) {$y=f(x)$};

    % Punkte
    \node[blue] (P1n) at (0.9,0.9) {$(x_1,y_1)$};
    \node[blue] (P1) at (0.2,0.9) {$\bullet$};

   \node[blue] (P2) at (2.7,2.7) {$\bullet$};
   \node[blue] (P3) at (1.7,2.6) {$\bullet$};
   \node[blue] (P4) at (0.7,2) {$\bullet$};

   \node[blue] (P5n) at (4.4,3.1) {$(x_n,y_n)$};
   \node[blue] (P5) at (3.7,3.1) {$\bullet$};

   % Koordinatensystem
   \draw[color=black,->] (-0.2,-0.1) -- (-0.2,3.1);
   \draw[color=black,->] (-0.3,0) -- (4,0);
\end{tikzpicture}
```



Zeichnen von Rechtecken und Verschiebung von Objekten

```
\begin{tikzpicture}
    \draw (0,0) -- (4,0) -- (4,2) -- (0,2) -- cycle;

    \begin{scope}[shift={(5,1)}]
      \draw[green, fill=blue, line width=3pt] (0,0) -- (4,0) -- (4,2) --
    \end{scope}
\end{tikzpicture}
```



Verwendung von Variablen

```
\begin{tikzpicture}
\pgfmathsetmacro{\PHI}{-15}
% Now use \PHI anywhere you want -15 to appear,
% can also be used in calculations like 2*\PHI
\def\x{10};

\draw[red] (0,4) -- (1-\PHI*0.5,4);
\draw[green] (0,2) -- (1+1/\x,2);
\draw[blue] (0,0) -- ({1+3*(\x/5+1)},0);
\end{tikzpicture}

\bigskip

%\usetikzlibrary{math} %needed tikz library

\begin{tikzpicture}
  \pgfmathsetmacro{\drehpunktx}{11.63815573}
  %Variables must be declared in a tikzmath environment but
```

```
  % can be used outside
  \tikzmath{\x1 = 1; \y1 =1;
  %computations are also possible
  \x2 = \x1 + 1; \y2 =\y1 +3; }
  \draw[->] (\x1, \y1)--(\x2, \y2);
\end{tikzpicture}
```

Verwendung von Punkten

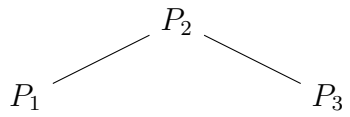
```
%\usetikzlibrary{backgrounds}  % is needed

  \begin{tikzpicture}
    \node [fill=gray!30] (P1) at (0,0) { $P_1$ };
    \node [fill=gray!30] (P2) at (2,0) { $P_2$ };
    \node [fill=gray!30] (P3) at (4,0) { $P_3$ };

    \begin{scope}[on background layer]
        \draw (P1) -- (P3);
    \end{scope}
  \end{tikzpicture}
```

P_1 ——— P_2 ——— P_3

Verwendung von Punkten

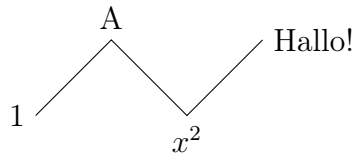
```
  \begin{tikzpicture}
    \node  (P1) at (0,0){$P_1$};
    \node  (P2) at (2,1){$P_2$};
    \node  (P3) at (4,0){$P_3$};
```

```
    \begin{scope}
        \draw (P1) -- (P2) -- (P3);
    \end{scope}
\end{tikzpicture}
```



Verwendung von Punkten

```
\begin{tikzpicture}
  \draw (0, 0) node[left]{1}
        -- ++(1, 1) node[above] {A}
        -- ++(1,-1) node[below] {$x^2$}
        -- ++(1, 1) node[right] {Hallo!};
\end{tikzpicture}
```



B. Kriterien für eine gutes LaTeX-Projekt

Ein guter Bericht zeichnet sich nicht nur durch den guten Inhalt aus, sondern erfüllt auch formale Aspekte. Die folgende Liste soll helfen, grundlegende Regeln einzuhalten. Vor der Abgabe sollten alle Punkte geprüft und abgehakt werden.

- ☐ Wird eine geeignete Verzeichnisstruktur verwendet?
- ☐ Wird eine geeignete Aufteilung in Dateien verwendet?
- ☐ Werden aussagekräftige Verzeichnis- und Dateinamen verwendet?
  - ☐ Werden Verzeichnis- und Dateinamen wie z.B. report oder Hausarbeit vermieden?
  - ☐ Werden für Bilder aussagekräftige Namen und nicht „image1“verwendet?
  - ☐ Sind die Verzeichnis- und Dateinamen nicht zu lang?
  - ☐ Werden Sonderzeichen und Freizeichen vermieden?
- ☐ Werden Befehle wie \newline und \\ vermieden?
- ☐ Werden die LaTeX-Dateien übersichtlich gestaltet?
  - ☐ Werden in den LaTeX-Dateien Einrückungen verwendet?
  - ☐ Werden Freizeilen eingefügt?
  - ☐ Wird im Header der Dateien das Projekt erwähnt?
  - ☐ Wird im Header der Dateien die Hauptquellen erwähnt?
  - ☐ Wird im Header der Dateien der Autor erwähnt?
- ☐ Werden alle notwendigen Dateien abgegeben?
- ☐ Werden die temporären Dateien gelöscht?
- ☐ Werden Grafiken selber erstellt?
- ☐ Werden die Quellen der Bilder angegeben?
- ☐ Wird mit dem Kommando \cite zitiert?
- ☐ Wird eine bib-Datei verwendet?

- ☐ Sind die Einträge der bib-Datei übersichtlich?
- ☐ Sind die Einträge der bib-Datei so editiert, dass sie korrekt dargestellt werden?
- ☐ Werden die korrekten Typen für die bib-Einträge verwendet?
- ☐ Werden aussagekräftige Schlüssel in den bib-Dateien verwendet?

Leider kann die Liste nicht vollständig sein, aber sie liefert einige Anhaltspunkte.